

日 本 国 特 許 庁
JAPAN PATENT OFFICE



別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日
Date of Application:

2000年 9月 7日

出 願 番 号
Application Number:

特願2000-270925

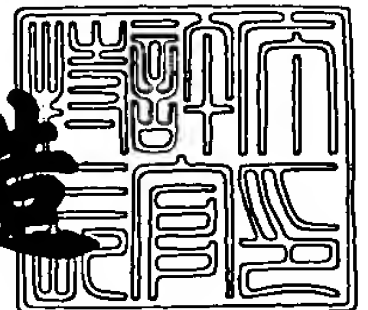
出 願 人
Applicant(s):

沖電気工業株式会社

2001年 8月 3日

特 許 庁 長 官
Commissioner,
Japan Patent Office

及 川 耕 造



出証番号 出証特2001-3068195

【書類名】 特許願

【整理番号】 SA003564

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/46

【発明者】

【住所又は居所】 東京都港区虎ノ門1丁目7番12号 沖電気工業株式会社
社内

【氏名】 内海 功朗

【特許出願人】

【識別番号】 000000295

【氏名又は名称】 沖電気工業株式会社

【代理人】

【識別番号】 100082050

【弁理士】

【氏名又は名称】 佐藤 幸男

【手数料の表示】

【予納台帳番号】 058104

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9100477

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 タスクプログラム制御システム

【特許請求の範囲】

【請求項 1】 複数のタスクの優先度と優先度別順位とを示す情報を記憶する記憶回路と、

前記記憶回路より、最も優先度の高いタスクの優先度を読み出す優先度読出回路と、

前記優先度読出回路で読み出した優先度の優先度別順位データを読み出す順位読出回路と、

前記優先度読出回路の値を保持する優先度レジスタと、

前記優先度読出回路で読み出した優先度内の順位の値を保持する順位レジスタと、

前記順位読出回路で読み出した優先度別順位データのうち、最も高い順位の値を前記順位レジスタに設定すると共に、当該順位レジスタの値が読み出された場合、そのタイミングで、当該読み出された値が、次に読み出す場合に最も低い順位となるよう設定する優先度別順位制御回路と、

前記優先度レジスタおよび順位レジスタの値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後は当該タスクが有するタスク自身を継続するか中止するかの値に基づき前記記憶回路の情報の更新を行うタスク実行手段とを備えたことを特徴とするタスクプログラム制御システム。

【請求項 2】 請求項 1 に記載のタスクプログラム制御システムにおいて、任意のタスクのアドレスが優先度レジスタおよび順位レジスタから読み出された場合、記憶回路の当該タスクに該当する情報をクリアするよう構成したことを特徴とするタスクプログラム制御システム。

【請求項 3】 請求項 1 または 2 に記載のタスクプログラム制御システムにおいて、

タスク実行手段からのリードアクセスの値をカウントするカウンタと、

特定の優先度に対応した所定値を有し、前記カウンタのカウント値が当該所定値を超えた場合に前記カウント値を出力するマスク回路と、

前記マスク回路からカウント値が出力された場合は、記憶回路より前記特定の優先度以下の優先度のうち最も優先度の高いタスクの優先度を読み出す特定優先度読出回路とを備えたことを特徴とするタスクプログラム制御システム。

【請求項4】 請求項1～3のいずれかに記載のタスクプログラム制御システムにおいて、

任意のタスクに対応したプログラムアドレスとデータアドレスとを示すプログラムテーブルとデータテーブルとを用意し、

タスク実行手段は、優先度レジスタおよび順位レジスタから任意のタスクのアドレスを読み出した場合は、前記プログラムテーブルとデータテーブルのアドレスに基づき、該当するタスクプログラムを実行し、かつ、データをアクセスするよう構成されたことを特徴とするタスクプログラム制御システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、例えばネットワークプロセッサにおけるプログラム制御を行うタスクプログラム制御システムに関するものである。

【0002】

【従来の技術】

従来、ハードウェアで行われていたネットワーク制御の処理をソフトウェアで行うネットワークプロセッサには、例えば、Lebel One(TM) IXP1200 Network Processor DataSheetに示されている技術があった。このネットワークを処理するプロセッサでは、四つのスレッド毎にレジスタバンクを持っているので、コンテキストの切替えはレジスタバンクの切替えで可能となる。但し、この構成ではレジスタバンクが4段しかないため、4より多いスレッドが実行される場合、コンテキストの退避が必要となり、多数のタスクを制御するのは困難である。

【0003】

また、コンテキストスイッチではなく、タスクを分割しポーリングすることによって、タスクのコンテキストスイッチのサイクル数を削減する方法として、例えば、特開平10-63517号公報「リアルタイム情報処理およびその装置」

に示されている方法があった。

【 0 0 0 4 】

【発明が解決しようとする課題】

しかしながら、上記レジスタバンクによる切替えの方法は、特化したプロセッサによる方法であり、汎用プロセッサではできない。また、レジスタバンクが四つのスレッドに対応しているため、4よりも数多いタスクを並列に実行すると、再びコンテキストの退避、回復の問題が発生する。

また、特開平 1 0 - 6 3 5 1 7 号公報に開示されている方法では、ポーリングのためにタスクを一定時間内に 1 回起動しなければならない、といった無駄なポーリング処理が必要となることや、タスクを分割しなければならないという問題点があった。

【 0 0 0 5 】

【課題を解決するための手段】

本発明は、前述の課題を解決するため次の構成を採用する。

＜構成 1＞

複数のタスクの優先度と優先度別順位とを示す情報を記憶する記憶回路と、記憶回路より、最も優先度の高いタスクの優先度を読み出す優先度読出回路と、優先度読出回路で読み出した優先度の優先度別順位データを読み出す順位読出回路と、優先度読出回路の値を保持する優先度レジスタと、優先度読出回路で読み出した優先度内の順位の値を保持する順位レジスタと、順位読出回路で読み出した優先度別順位データのうち、最も高い順位の値を順位レジスタに設定すると共に、順位レジスタの値が読み出された場合、そのタイミングで、読み出された値が、次に読み出す場合に最も低い順位となるよう設定する優先度別順位制御回路と、優先度レジスタおよび順位レジスタの値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後はタスクが有するタスク自身を継続するか中止するかに基づき記憶回路の情報の更新を行うタスク実行手段とを備えたことを特徴とするタスクプログラム制御システム。

【 0 0 0 6 】

＜構成 2＞

構成 1 に記載のタスクプログラム制御システムにおいて、任意のタスクのアドレスが優先度レジスタおよび順位レジスタから読み出された場合、記憶回路のそのタスクに該当する情報をクリアするよう構成したことを特徴とするタスクプログラム制御システム。

【 0 0 0 7 】

〈構成 3〉

構成 1 または 2 に記載のタスクプログラム制御システムにおいて、タスク実行手段からのリードアクセスの値をカウントするカウンタと、特定の優先度に対応した所定値を有し、カウンタのカウント値が所定値を超えた場合にカウント値を出力するマスク回路と、マスク回路からカウント値が出力された場合は、記憶回路より特定の優先度以下の優先度のうち最も優先度の高いタスクの優先度を読み出す特定優先度読出回路とを備えたことを特徴とするタスクプログラム制御システム。

【 0 0 0 8 】

〈構成 4〉

構成 1 ～ 3 のいずれかに記載のタスクプログラム制御システムにおいて、任意のタスクに対応したプログラムアドレスとデータアドレスとを示すプログラムテーブルとデータテーブルとを用意し、タスク実行手段は、優先度レジスタおよび順位レジスタから任意のタスクのアドレスを読み出した場合は、プログラムテーブルとデータテーブルのアドレスに基づき、該当するタスクプログラムを実行し、かつ、データをアクセスするよう構成されたことを特徴とするタスクプログラム制御システム。

【 0 0 0 9 】

【発明の実施の形態】

以下、本発明の実施の形態を具体例を用いて詳細に説明する。

本発明は、コンテキストスイッチのサイクル数を削減するための、状態遷移で制御されたタスクプログラムに対して高速にタスクを切替えるようにしたシステムである。

【 0 0 1 0 】

《具体例 1》

〈構成〉

図 1 は、本発明のタスクプログラム制御システムの具体例 1 を示す構成図であるが、これに先立ち、タスクプログラムの基本的な制御方法について説明する。

【0011】

図 2 は、本発明の具体例 1 の全体的な構成図である。

図示の装置は、CPU 100、タスクスケジューラ 110、メインメモリ 120、バス 130 からなる。CPU 100 は、タスクスケジューラ 110 の決定に従い、メインメモリ 120 に格納されている各種のプログラムを実行するプロセッサであり、本具体例では特定の CPU は想定していない。

タスクスケジューラ 110 は、図 1 に示されている回路で構成され、CPU 100 に対して次に実行させるタスクの番号を生成するための機能を有している。

メインメモリ 120 は、各種のプログラムやデータを格納し、また、CPU 100 の作業領域として用いられるメモリであり、プログラムテーブル 121、メインルーチン 122、各種タスクプログラム 123 を格納している。プログラムテーブル 121、メインルーチン 122 および各種タスクプログラム 123 については後述する。

また、バス 130 は、CPU 100、タスクスケジューラ 110、メインメモリ 120 を接続するシステムバスである。

【0012】

先ず、CPU 100 で実行されるタスクのプログラムの構成と、そのプログラムを起動させるメインルーチン 122 の構成について説明する。

例えば、ネットワークでなされるプロトコル処理は、物理層以外の下位レイヤの処理においても、次のようなシーケンスとみなして処理が可能である。

図 3 は、典型的なプロトコルシーケンスの処理例の説明図である。

プロトコルの処理上、各シーケンスの処理内容は単純であるが、他のシーケンスと並列に動作し、通信しながら実行させることによって、複雑な制御が可能である。

図 4 は、図 3 のプロトコルの処理を関数で表した説明図である。

図中、proc0～2は、各処理の名札に相当する。また、recieve(chanel0,data);は通信路chanel0からデータdataを受けることを示す。send(chanel1,data);は通信路chanel1へデータdataを転送することを示す。return;は状態を開始状態に設定することを示す。

【 0 0 1 3 】

ところで、図3に示すシーケンス処理において、通常、そのシーケンスだけに閉じている処理に比べて、受信処理や送信処理はサイクル数がかかる。これは受信や送信の相手のシーケンサが用意できるまで、実行が待たされるためである。この場合、リアルタイムOS上では、別のタスクを実行する制御が取られる。

【 0 0 1 4 】

図5は、本発明の具体例1で実行されるタスクプログラムの説明図であり、(a)はフローチャート、(b)は、フローチャートの内容をC言語で記述したものである。

図3のように状態を持ったシーケンス処理は状態に応じた分岐命令と次の状態設定の形に書き換えできる。

ステップS1は、そのタスクの状態を保持する変数である。

ステップS2は、状態変数に応じた分岐命令である。

ステップS3は、状態に応じた処理内容で、構成は、ステップS3-1～ステップS3-4に分かれている。

即ち、ステップS3-1は、状態を表す数値で、ステップS2の分岐命令は、この値によって分岐する。

ステップS3-2は、状態に応じた処理である。

ステップS3-3は、次に実行される時の状態を設定する。

ステップS3-4は、リターン命令で、タスクから抜け出る。このときに、タスクスケジューラ110によって再度実行されるときは“1”を、待ち状態あるいは終了するときには“0”を返す。

【 0 0 1 5 】

図5(b)中のコメント(//)は、図5(a)中のステップS**に対応している(**は任意の数値を表す)。

図 5 (b) 中の case1:にある get(chanel0,data)は、図 4 の recieve(chanel0,data)に対応する。違いは recieve(chanel0,data)が、data が chanel0 に来るまで待つことに対して、get(chanel0,data)は、既に chanel0 にあるデータ data をリードするだけでよく、ウェイトは発生しない。図 5 (b) 中の構成では、待つ処理を明示的に示さず、コメント S3-4 の return(0)によって、処理をメインルーチン 122 に返すことにしている。

【 0 0 1 6 】

図 6 は、タスク呼出しを行うメインルーチン処理の説明図であり、(a) はフローチャート、(b) は (a) の処理を C 言語で示した説明図である。

先ず、ステップ S 1 1 で、特定のアドレスに割り付けられたタスクスケジューラアドレス (図 6 (b) の MACRO_SCHEDULER_ADR) をレジスタ (図 6 (b) の sp) に設定する。次に、ステップ S 1 2 で、タスクスケジューラ 1 1 0 から次に実行するタスク番号 (図 6 (b) の NUM) を取り出す。タスクスケジューラ 1 1 0 は、いつでも次に実行するタスク番号を生成する。

次に、ステップ S 1 3 で、そのタスク番号のプログラム (図 6 (b) における func [NUM] ()) を実行する。

そして、ステップ S 1 4 で、タスクプログラムの実行の返り値をタスク番号に対応した状態 (図 6 (b) の state [NUM]) に反映させる。

タスクの状態は、次に説明するタスクスケジューラ 1 1 0 内のビットマトリクスに相当する。また、タスク番号は、ビットマトリクスのアドレスに対応している。そして、ステップ S 1 2 ～ステップ S 1 4 を無限ループで実行する。

【 0 0 1 7 】

次に、図 1 に示されているタスクスケジューラ 1 1 0 の構成について説明する。

図 1 に示すシステムは、記憶回路であるマトリクス回路 1、優先度読出回路である優先度プライオリティエンコーダ (PE) 2、デコーダ (DECODE) 3、ドライバ (DR) 4、順位読出回路であるマルチプレクサ (MUX) 5、優先度別順位制御回路 6 (インバータ (IV) 7、AND 回路 8, 9、プライオリティエンコーダ (PE) 10, 11、OR 回路 12、マルチプレクサ (MUX) 1

3、優先度別順位レジスタ14、マルチプレクサ(MUX)15、デコーダ(DEC)16、減算器17)、優先度レジスタ(PRI R)18、順位レジスタ(ODRR)19、タスク実行手段20からなる。

【0018】

マトリクス回路1は8×8ビットマトリクスであり、図面縦方向は優先度、横方向は同優先度での順位を表す。尚、上の方が優先度が高く、また、左の方が同一優先度での順位が高い。本具体例では、8通りの優先度で、各優先度で8通りの順位を持つ。但し、本発明はこのような優先度および順位の数に限定されるものではない。このマトリクス回路1では、各タスク毎に1ビットの状態を持つ。ここで、0は待ち状態または休止状態、1は実行可能状態である。優先度の数が大きく、また、同優先度で順位の高い、実行可能状態にあるビットに割り付けられたタスクがCPU100で実行されるよう構成されている。

【0019】

優先度プライオリティエンコーダ2は、各優先度毎に実行可能状態になっているビットの論理和(OR)を取り、その論理和に対してプライオリティエンコードして、実行可能状態にある最も高い優先度を検出する機能を有している。出力は3ビットで、優先度7の行にビットが立っていた場合出力は111となり、どの優先度のラインにもビットが立っていない場合出力は000となる。

【0020】

デコーダ3は、CPU100からのアクセス要求とアドレスから優先度レジスタ18と順位レジスタ19のデータをデータバスに出力するためのドライブ信号を生成する機能を有している。また、このドライブ信号は優先度別順位レジスタ14の値を更新するのにも使用される。ドライバ4はデータバスへのドライバである。

マルチプレクサ5はマトリクス回路1から優先度プライオリティエンコーダ2が指定した優先度のラインビット(優先度別順位データ)を取り出すためのマルチプレクサである。

【0021】

優先度別順位制御回路6は、マルチプレクサ5で読み出した優先度別順位デー

タのうち、最も高い順位の値を順位レジスタ 1 9 に設定すると共に、この順位レジスタ 1 9 の値が読み出された場合、そのタイミングで、読み出された値が、次に読み出す場合に最も低い順位となるよう設定する機能を有しており、インバータ 7 ～減算器 1 7 から構成されている。

インバータ 7 は、デコーダ 1 6 の出力を反転する回路である。AND 回路 8 は、インバータ 7 で反転したデコーダ 1 6 の出力とマルチプレクサ 5 の出力との論理積 (AND) を取る回路である。また、AND 回路 9 は、デコーダ 1 6 の出力とマルチプレクサ 5 の出力との論理積 (AND) を取る回路である。

【 0 0 2 2 】

プライオリティエンコーダ 1 0, 1 1 は、それぞれ AND 回路 8, 9 の出力をプライオリティエンコードして最も順位の高いビットを求めるためのプライオリティエンコーダである。即ち、プライオリティエンコーダ 1 0 は、デコーダ 1 6 から出力されるマスク信号に基づくビットのマスク部分で最も順位の高いビットを求めるプライオリティエンコーダであり、プライオリティエンコーダ 1 1 は、マスクされていない部分で最も順位の高いビットを求めるプライオリティエンコーダである。

OR 回路 1 2 は、AND 回路 9 の出力の論理和 (OR) を取る回路である。AND 回路 9 の出力に 1 ビットでも 1 が立っていた場合は 1 が出力されることになる。また、マルチプレクサ 1 3 は、二つのプライオリティエンコーダ 1 0, 1 1 からの出力を選択するマルチプレクサである。即ち、マスクされていない入力に対するプライオリティエンコーダ出力が常に優先して選択される。

【 0 0 2 3 】

優先度別順位レジスタ 1 4 は、優先度別にそのレベルでのタスクが待ち状態に切り替わった時にその順位を保持するための回路である。尚、この保持タイミングは上述したようにデコーダ 3 からのドライブ信号がアサートされた時である。マルチプレクサ 1 5 は、優先度プライオリティエンコーダ 2 出力に応じて優先度別順位レジスタ 1 4 からの出力を選択するマルチプレクサである。デコーダ 1 6 は、マルチプレクサ 1 5 出力を 3 ビットから 8 ビットにデコードするためのデコーダである。即ち、デコーダ 1 6 は、優先度別順位レジスタ 1 4 に保持された順

位に基づくマスク信号を発生する。減算器 1 7 は、順位レジスタ 1 9 の値を 1 デクリメントさせる減算器である。

【 0 0 2 4 】

優先度レジスタ 1 8 は、優先度プライオリティエンコーダ 2 の出力であるマトリクス回路 1 の優先度を保持するレジスタである。また、順位レジスタ 1 9 は、マルチプレクサ 1 3 の出力である優先度別順位の値を保持するレジスタである。

【 0 0 2 5 】

タスク実行手段 2 0 は、CPU 1 0 0 がメインルーチン 1 2 2 を実行することで実現される機能であり、優先度レジスタ 1 8 および順位レジスタ 1 9 の値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後はこのタスクが有するタスク自身を継続するか中止するかの値に基づきマトリクス回路 1 の情報の更新を行うものである。

【 0 0 2 6 】

〈動作〉

動作の一例として三つのタスクプログラムの動作する様子を示して、図 5 のタスクプログラム、図 6 のメインルーチンおよび図 1 の動作の説明を行う。

図 7 は、タスクの初期割付状況の説明図である。

図中、(a) は各タスクのアドレスを示している。図示のように、各タスクのアドレスは、6 ビットの数値で表す。最初の b はこれらのアドレスが 2 値表示であることを示す記号である。また、6 ビットの“0”または“1”の中間に設けたアンダースコアは優先度と優先度別順位とを区別できるようにした表示で、実質的な意味はない。

【 0 0 2 7 】

図中、(b) はマトリクス回路 1 におけるビット状態を示している。例えば、task0 はアドレスとして b110_101 を持ち、8 × 8 ビットマトリクス上では 6 行 5 列目に相当する場所に 1 が立っている。同じく task1 はアドレスとして b011_110 を持ち、8 × 8 ビットマトリクス上では 3 行 6 列目に相当する場所に 1 が立っている。また、task2 はアドレスとして b011_011 を持ち、8 × 8 ビットマトリクス上では 3 行 3 列目に相当する場所に 1 が立っている。これらの設定は、図 6 (b

）に示すメインルーチンプログラムが実行される前の初期設定で設定される。尚、マトリクス回路 1 において、b000-000は予約されている。

【 0 0 2 8 】

図 8 は、各タスクプログラムの説明図であり、（a）がtask0、（b）がtask1、（c）がtask2のプログラムを示している。

尚、各タスクプログラムの制御構造は図 5 （b）と同様である。

【 0 0 2 9 】

以降、タスクスケジューラ 1 1 0 と三つのタスクプログラムがどのように動作するかを説明する。尚、ハードウェアとソフトウェアの説明を区別するため、ハードウェアの手順にはステップ* *_Hで、ソフトウェアの手順にはステップ* *_Sを表記するものとする（**は任意の数値を表す）。

【 0 0 3 0 】

[ステップ 1 _H]

マトリクス回路 1 から優先度プライオリティエンコーダ 2 は、データを取り出す。図 7 に示した通り、1 が立っている最大のラインは 6 行なので 6 （=b110）を出力する。

[ステップ 2 _H]

マルチプレクサ 5 より、優先度プライオリティエンコーダ 2 出力に従って 6 行目のビット列がマトリクス回路 1 より選択される。

[ステップ 3 _H]

優先度別順位レジスタ 1 4 からマルチプレクサ 1 5 によって 6 行目の優先度順位を取り出す。通常、初期状態ではこれらの優先度順位はb111である。尚、b111は、十進法の 7 を示し、本具体例では最も優先度が高いことを表している。

【 0 0 3 1 】

[ステップ 4 _H]

デコーダ 1 6 でb111 （= 7）に相当するパターンを 8 ビットのb11111111にデコードする。尚、デコードは、例えば、b000 （= 0）はb00000001に、b001 （= 1）は、b00000011のように行うものとする。

[ステップ 5 _H]

このデコードされたデータは、AND回路8とAND回路9でマルチプレクサ5から出力されたデータとAND論理が取られる。AND回路8ではインバータ7によって反転されたデータとAND論理が取られることになる。即ち、AND回路8では全てのビットがマスクされたデータ (b00000000) とマルチプレクサ5のデータとのAND論理が取られ、AND回路9では全てのビットがマスクされていないデータ (b11111111) とマルチプレクサ5のデータとのAND論理が取られる。その結果、AND回路8の出力はb00000000、AND回路9の出力はb00100000となる。

【 0 0 3 2 】

[ステップ6__H]

プライオリティエンコーダ10, 11の出力は、AND回路8, 9の出力を通して、b000とb101である。また、OR回路12でAND回路9の出力においていずれかのビットが1であることが判明する。

[ステップ7__H]

マルチプレクサ13でOR回路12の値に従って、プライオリティエンコーダ10, 11のいずれかの出力を選択する。OR回路12の出力が1であるため、プライオリティエンコーダ11の出力が選択され、その出力はb101である。

[ステップ8__H]

以上の結果により、優先度レジスタ18と順位レジスタ19は、それぞれb110とb101が入る。これは、task0のアドレス値が上位と下位に分かれて入っていることに相当する。

【 0 0 3 3 】

[ステップ9__S]

初期設定が終了して、図6(b)の(S11)で、CPU100は図示しないレジスタにタスクスケジューラアドレスを設定する。そして無限ループに入り、(S12)でタスクスケジューラ110から上記ステップ8__Hで得られたデータを取り出す。

【 0 0 3 4 】

[ステップ10__H]

デコーダ 3 で、ステップ 9 __S のリードアクセスをデコードして、ドライバ 4 を介してデータバス上に優先度レジスタ 1 8 と順位レジスタ 1 9 の値を出力する。

[ステップ 1 1 __H]

デコーダ 3 からのドライブ信号は、優先度別順位レジスタ 1 4 の設定信号にもなっているので、順位レジスタ 1 9 の値から 1 減少したデータ b100 が優先度レジスタ 1 8 の相当するレベル、ここでは優先度 6 に対応するレジスタ L 6 に設定される。

【 0 0 3 5 】

[ステップ 1 2 __S]

図 6 (b) のメインルーチンでリードしてきたデータは変数 NUM に設定される。そして、この NUM に対応する配列の関数を実行する (図 6 (b) における (S 1 3) および図 6 (a) におけるステップ S 1 3)。

【 0 0 3 6 】

図 9 は、メインメモリ 1 2 0 上でのプログラムテーブル 1 2 1 の割付状態を示す説明図である。

図示のように、メモリ上にプログラムテーブルとして配列 func が割り付けられ、task0, task1, task2 の番号に相当するアドレスに、task0, task1, task2 の実行開始アドレスが割り付けられている。従って、図 6 (b) の (S 1 3) において、task0 番地に分岐し、実行することに相当する。

【 0 0 3 7 】

[ステップ 1 3 __S]

図 8 (a) の task0 が実行され、状態変数 sta0 の値が初期値で 0 であることにより、case0 に対応する命令が実行される (図 8 中の // 1 3 __S) この部分で状態変数 1 に更新する (sta0=1)。更に 0 が返される (return(0))。

[ステップ 1 4 __S]

図 6 (b) で、リターン値は配列 state の NUM 番地に相当するところに 0 が書き込まれる。配列 state はマトリクス回路 1 の 8 × 8 ビットマトリクスの行と列に相当する。よって、マトリクス回路 1 での NUM (=b110_101) に対応するビット

が0にクリアされる。

〔ステップ15__H〕

マトリクス回路1の6行5列がクリアされたので、1が立っているのは3行の2ビットだけである。従って、優先度プライオリティエンコーダ2の出力は3となる。よって、マルチプレクサ5は3行目のビット列を取り出す。また、優先度別順位レジスタ14の初期値はいずれもb111(=7)なので、上記のステップ4__H～ステップ8__Hと同様に、優先度レジスタ18と順位レジスタ19には、task1のアドレスであるb011とb110がそれぞれ入る。

【0038】

〔ステップ16__S〕

メインルーチンの無限ループによって、再度、タスクスケジューラ110から次に実行すべき番号を取り出す(図6(a)、(b)のステップS12および／／(S12))。

〔ステップ17__H〕

タスクスケジューラ110からのリードと共に、優先度別順位レジスタ14の優先度3に対応するレジスタには順位レジスタ19の値から1減少させた値101が入る。データバスにはb011110が出力される。

〔ステップ18__S〕

図6(b)において、NUMにはb011110が設定され、関数の配列funcからアドレスb011110に相当するtask1を実行する(図6(a)、(b)のステップS13および／／(S13))。

〔ステップ19__S〕

図8(b)に示すtask1が実行され、状態変数sta1の値が初期値で0であることにより、case0に対応する命令が実行される(図8の／／19__S)。この部分で状態変数を1に更新する(sta=1)。更に1が返される(return(1))。

〔ステップ20__S〕

図6(b)において、リターン値は配列stateのNUM番地に相当するところに1が書き込まれる。配列stateはマトリクス回路1の行と列に相当する。よって、マトリクス回路1でのNUM(=b011_101)に対応するビットが1に再設定される

【 0 0 3 9 】

[ステップ 2 1 __H]

マトリクス回路 1 の 6 行 5 列がクリアされたので、1 が立っているのは 3 行の 2 ビットだけである。従って、優先度プライオリティエンコーダ 2 の出力は 3 となる。よって、マルチプレクサ 5 は 3 行目のビット列を取り出す。また、優先度別順位レジスタ 1 4 の優先度 3 に相当するレジスタは 101 に更新されているので、デコーダ 1 6 には b00111111 が出力される。

【 0 0 4 0 】

[ステップ 2 2 __H]

AND 回路 8, 9 でそれぞれデコーダ 1 6 出力 b00111111 とマルチプレクサ 5 出力 01001000 の論理積を取ると、それぞれ b01000000 と b00001000 が出力される。OR 回路 1 2 の出力が 1 となることにより、AND 回路 9 の出力を入力とするプライオリティエンコーダ 1 1 の出力が選択され、順位レジスタ 1 9 には b011 が入る。

このような構成で、一度実行されたタスクは次に実行される場合、同優先度内では最後に実行されるようになる。即ち、6 ビット目の順位のタスクが実行された後は、優先度別順位レジスタ 1 4 の値が b101 (= 5) となり、6、7 ビット目のデータがマスクされるマスク信号がデコーダ 1 6 から出力される。その結果、プライオリティエンコーダ 1 0 の値は b110 であり、プライオリティエンコーダ 1 1 の値は b011 であるが、AND 回路 9 の出力に “1” が立っているため、プライオリティエンコーダ 1 1 の出力が選択されることになる。

【 0 0 4 1 】

[ステップ 2 3 __S]

メインルーチンの無限ループによって、再度、タスクスケジューラ 1 1 0 から次に実行すべき番号を取り出す (図 6 (a)、(b) におけるステップ S 1 2 および // (S 1 2))。

[ステップ 2 4 __H]

タスクスケジューラ 1 1 0 からのリードと共に、優先度別順位レジスタ 1 4 の

優先度 3 に対応するレジスタには順位レジスタ 1 9 の値から 1 減少させた値 010 が入る。データバスには b011011 が出力される。

[ステップ 2 5 __ S]

図 6 (b) において、NUM には b011011 が設定され、関数の配列 func からアドレス b011011 に相当する task2 を実行する (図 6 (a)、(b) におけるステップ S 1 3 および // (S 1 3))。

[ステップ 2 6 __ S]

図 8 の (c) に示す task2 が実行され、状態変数 sta2 の値が初期値で 0 であることにより、case0 に対応する命令が実行される。“state [task0_id]” の task0_id は task0 のアドレス b110101 に相当し、1 を書き込むということは、マトリクス回路 1 の b110 (= 6) 行、b101 (= 5) 列に 1 を書き込むことに相当する。

図 8 の (c) 中の // 2 6 __ S の部分で状態変数 1 に更新する (sta2=1)。更に 1 が返される (return(1))。

【0 0 4 2】

[ステップ 2 7 __ S]

図 6 (b) において、リターン値は配列 state の NUM 番地に相当するところに 1 が書き込まれる。配列 state はマトリクス回路 1 の行と列に相当する。よって、マトリクス回路 1 での NUM (= b011_011) に対応するビットが 1 に再設定される。

[ステップ 2 8 __ H]

マトリクス回路 1 の 6 行目に 1 が立ったため、優先度プライオリティエンコーダ 2 はデータを取り出し、図 7 に示した通り、1 が立っている最大のラインは 6 行なので 6 を出力する。これ以降は、上述したステップ 4 __ H ~ ステップ 8 __ H と同じ処理をたどる。

【0 0 4 3】

[ステップ 2 9 __ S]

図 6 の (b) のメインルーチンに示す無限ループによって、再度タスクスケジューラ 1 1 0 から次に実行すべき番号を取り出す (図 6 (a)、(b) におけるステップ S 1 2 および // (S 1 2))。NUM には b110101 が設定され、関数の配

列funcからアドレスb110101に相当するtask0を実行する（図6（a）、（b）におけるステップS13および／／（S13））。

〔ステップ30__S〕

図8の（a）の状態変数sta0は1なので、case1以下の命令が実行される。状態変数sta0は0に設定し、return(0)で0を返す。

〔ステップ31__H〕

マトリクス回路1の6行5列がクリアされたので、1が立っているのは3行の2ビットだけである。従って優先度プライオリティエンコーダ2の出力は3となる。よって、マルチプレクサ5は3行目のビット列を取り出す。また、優先度別順位レジスタ14の優先度3のレジスタ値はb010（=2）なので、今度は、AND回路8, 9の出力は、それぞれ、b10010000とb00000000となる。従って、OR回路12の出力は0となり、マルチプレクサ13の出力はプライオリティエンコーダ10の出力b110で、これはtask1のアドレスの下位ビットである。よって、task1が実行されることになる。

〔ステップ32__S〕

以降、task1が起動し、続いてtask2が起動する。

【0044】

図10は、このようなタスクの実行順序を示す説明図である。

このように、マトリクス回路1の優先度のビット指定に従ってタスクが交互に実行される。また、スケジュール部分をハードウェアで、実行起動部分をソフトウェアでやっているのので、どのようなCPU100でも適用が可能である。

【0045】

〈効果〉

以上のように、具体例1によれば、マトリクス回路1や優先度別順位制御回路6等からなるタスクスケジューラ110で、次に実行するタスクを決定し、かつ、各タスクは、自身を継続するか中止するかの情報を持ち、タスクを実行した場合は、この情報に基づいてマトリクス回路1の状態を更新するようにしたので、タスク切替えを高速に行うことができる。例えば、本具体例で示したC言語のプログラムおよび各タスクの状態名札への分岐部分はアセンブラプログラムでそれ

ぞれ 5 命令程度である。よって、実行中のタスクプログラムから次に優先順位の高いタスクプログラムへはデータアクセス、分岐命令によるロスを考慮しても 20 サイクル以内で可能となる。汎用プロセッサにおいて、コンテキストスイッチが最低でも 100 サイクル程度必要とされることを考えると、極めて高速な処理方法である。

また、本具体例では CPU 100 の構造に対して限定がなく、どのような CPU あるいは DSP でも適用可能である。

【 0 0 4 6 】

《具体例 2》

〈構成〉

図 1 1 は、具体例 2 の全体構成図である。

上述した具体例 1 では CPU 100 が一つの場合であったのに対し、具体例 2 では複数の CPU 100 (100-1、100-2) が同じバス 130 上に接続されている場合である。この場合のタスクスケジューラ 110 a の構成は次のようになっている。

【 0 0 4 7 】

図 1 2 は、具体例 2 におけるタスクスケジューラ 110 a の要部構成図である。

基本的な構成は具体例 1 と同様であるが、タスクスケジューラ 110 a をリードする、即ち、優先度レジスタ 18、順位レジスタ 19 (図 1 2 ではこれらの図示は省略している) を読む制御信号によって、そのアドレスデータに基づいてクリアするようになっている。他の各構成は図 1 に示した構成と同様であるため、ここでの説明は省略する。

【 0 0 4 8 】

〈動作〉

具体例 1 のタスクの例を用いて説明する。

まず、CPU 100-1 がバス権を取得してタスクスケジューラ 110 a にアクセスすると、具体例 1 と同様に task0 のアドレス値が得られる。CPU 100-1 はそのまま task0 を実行する。ここで、具体例 2 では、優先度レジスタ 18

および順位レジスタ 19 の値を読み出すと同時に task0 のアドレス値に対応するビットがクリアされる。

次に、CPU 100-1 が task0 を実行中、CPU 100-2 がタスクスケジューラ 110a にアクセスしたとすると、task0 のビットはクリアされているので、task1 のアドレス値が読み出され、task1 が実行される。

【0049】

このように、最も優先度が高いタスクに対応したタスクアドレスを読み出す度に、マトリクス回路 1 上のそのタスクアドレスに対応するビットをクリアすることで、複数の CPU 100 が同一のタスクを実行するということがなくなり、複数の CPU 100 であっても、優先順位別に別々のタスクを実行することができる。

【0050】

〈効果〉

以上のように具体例 2 によれば、任意のタスクのアドレスが読み出された場合は、マトリクス回路 1 におけるそのタスクのアドレス値をクリアするようにしたので、複数の CPU が存在した場合でも、重複することなくタスクを実行することができる。

【0051】

《具体例 3》

複数のタスクにおいて、優先度の高いタスクが多く割り付けられている場合、優先度の低いタスクが全く実行されなくなるといった事態が生じる。これは一般のリアルタイム OS 上でも発生している。これを解決するため、具体例 3 では、低い優先度のタスクでも周期が低い形で実行できるようにしている。

【0052】

〈構成〉

図 13 は、具体例 3 におけるタスクスケジューラの要部構成図である。

図では、具体例 1 と異なっている部分を示しており、マトリクス回路 1 および優先度プライオリティエンコーダ 2 は具体例 1 と同様である。具体例 3 では、更に、カウンタ (COUNTER) 21、マスク回路 (MSK) 22、特定優先度

読出回路 23 (AND 回路 24、プライオリティエンコーダ 25、OR 回路 26、マルチプレクサ 27) が追加されている。他の各構成は具体例 1 と同様であるため、その図示および説明は省略する。

【0053】

カウンタ 21 はリードアクセスによってカウントアップするカウンタである。また、OR 回路 25 の出力が “1” となってリードアクセスが行われると “0” にクリアされる。この例では、図面の上から下に向かってカウンタ値が増えるものとする。マスク回路 22 は、カウンタ 21 の出力のうち、あるビットの範囲だけマスクするマスク回路である。即ち、特定の優先度に対応した所定値を有し、カウンタ 21 の値が所定値を超えた場合にそのカウント値を出力するよう構成されている。つまり、優先度が高いものは比較的実行する可能性が高いので、それをマスクして、優先度の低いものが実行できるようにさせるための回路である。特定優先度読出回路 23 は、マスク回路 22 からカウント値が出力された場合は、マトリクス回路 1 より特定の優先度のうち最も優先度の高いタスクの優先度を読み出す機能を有しており、AND 回路 24 ~ マルチプレクサ 27 で構成されている。

【0054】

AND 回路 24 は、マトリクス回路 1 の優先度出力とマスク回路 22 出力との AND 演算を行う AND 回路である。プライオリティエンコーダ 25 は、AND 回路 24 の出力のうち最も優先度の高い優先度を取り出すものであり、優先度プライオリティエンコーダ 2 と同様 3 ビットの信号を OR 回路 26 およびマルチプレクサ 27 に出力する。OR 回路 26 は、プライオリティエンコーダ 25 出力に、1 であるビットがあるかどうかを判定し、あった場合は “1” を出力する OR 回路である。また、OR 回路 26 の出力はカウンタ 21 のクリアイネーブルに接続されている。マルチプレクサ 27 は優先度プライオリティエンコーダ 2 またはプライオリティエンコーダ 25 のいずれかの出力を選択するセレクタであり、OR 回路 26 の出力が 1 であった場合にプライオリティエンコーダ 25 の出力を選択するよう構成されている。

尚、図示省略しているが、具体例 3 のマルチプレクサ 5 は、マルチプレクサ 2

7 の出力が与えられ、この出力に基づいて優先度別順位を読み出すよう構成されている。

【 0 0 5 5 】

＜動作＞

図 1 4 は、動作の一例を示すための説明図である。

マトリクス回路 1 には図示のようにビットが割り当てられているとする。このうち、6 行目に割り当てられているタスクばかり実行していて、2 行目に割り当てられたタスクは実行がされたことがないとする。

【 0 0 5 6 】

このような状態で、優先度プライオリティエンコーダ 2 の出力は 6 である。ここで、カウンタ 2 1 の値が b00100000 になったとする（図では、上位がマトリクス回路 1 のビットマトリクスの 0 行目、下位が 7 行目に対応する）。また、マスク回路 2 2 では、上位 3 ビットのみ 1 とし、それ以外の下位は 0 にマスクされるよう設定されているとする。よって、マスク回路 2 2 の出力は b00100000 となる。

AND 回路 2 4 は、マトリクス回路 1 のビット出力 b10110010 とマスク回路 2 2 出力 b00100000 とを AND 演算し、b00100000 を出力する。その結果、プライオリティエンコーダ 2 5 は b010 (= 2) を出力する。これにより、OR 回路 2 6 の出力が 1 となり、マルチプレクサ 2 7 はプライオリティエンコーダ 2 5 の出力 b010 を選択して出力する。

【 0 0 5 7 】

即ち、カウンタ 2 1 が 6 4 回リードアクセスになったときには、優先的に優先度 2 のタスクが起動されることになる。また、OR 回路 2 6 の出力により次のタスクのリード時にはカウンタ 2 1 がクリアされる。

また、上記例では、優先度 2 のタスクを優先的に起動する場合について説明したが、これ以外の優先度でもマスク回路 2 2 のマスクビットの設定によって任意の優先度のタスクを起動させることができる。

【 0 0 5 8 】

＜効果＞

以上のように、具体例 3 によれば、リードアクセスの値をカウントするカウンタ 2 1 と、このカウンタ 2 1 出力を、特定の優先度に対応した値でマスクするマスク回路 2 2 と、特定の優先度に対応したカウント値になった場合は、マトリクス回路 1 から、その優先度以下で最も優先度の高い優先度を読み出す特定優先度読出回路 2 3 を設けたので、具体例 1 の効果に加えて、優先度の低いところに割り付けられたタスクでも周期的に選択されるので、優先度の低いタスクでも起動させることができるという効果がある。

【 0 0 5 9 】

《具体例 4》

具体例 4 では、同じデータ構成であるが異なったデータに対して同じ処理を行う場合に効果が上がるようにしたものである。例えば、このような例としては A T M セルデータの処理がある。本具体例は主としてソフトウェア構成にかかわるものである。

【 0 0 6 0 】

〈構成〉

図 1 5 は、具体例 4 の全体構成図である。

図の装置は、具体例 1、2 と比べてメインメモリ 1 2 0 内の構成が異なっている。メインメモリ 1 2 0 内には、プログラムテーブル 1 2 1 に加えてデータテーブル 1 2 4 が設けられ、かつ、メインルーチン 1 2 2 a と各種タスクプログラム 1 2 3 を有している。

【 0 0 6 1 】

図 1 6 は、プログラムテーブル 1 2 1 とデータテーブル 1 2 4 の構成説明図である。

具体例 1 の説明でも示した通り、タスクスケジューラ 1 1 0 は次に実行するタスク番号を C P U 1 0 0 に渡し、C P U 1 0 0 はメインメモリ 1 2 0 上のプログラムテーブル 1 2 1 から分岐するプログラム先頭アドレスをアクセスして、そのアドレスに分岐することにより該当するタスクを実行していた。

これと同じく、次に実行するタスク番号からデータテーブル 1 2 4 をアクセスしてやり、そのデータに対して処理する構成をとる。図 1 6 の (a) に示したプ

プログラムテーブル 1 2 1 では、タスクスケジューラ 1 1 0 が取る得るタスク番号に対応したアドレスにそのプログラム先頭アドレスが設定されている。また、(b) に示したデータテーブル 1 2 4 では、同様のアドレス割付でデータがテーブルに設定されている。

【 0 0 6 2 】

これにより、タスク実行手段 2 0 の機能を実現するメインルーチン 1 2 2 a では、マトリクス回路 1 より特定のタスク番号を読み出した場合、プログラムテーブル 1 2 1 とデータテーブル 1 2 4 のアドレスに基づき、そのプログラムを実行し、かつ、データにアクセスするよう構成されている。他の各構成は具体例 1 と同様であるため、ここでの説明は省略する。

【 0 0 6 3 】

〈動作〉

図 1 7 は、具体例 4 で実行されるタスクプログラムの構造を示す説明図であり、(a) はそのフローチャート、(b) は、フローチャートの内容を C 言語で記述したものである。

図 1 7 に示す具体例 4 のタスクプログラムと、図 5 で示した具体例 1 のタスクプログラムとの違いは、具体例 4 のタスクプログラムでは、状態変数の宣言 (図 5 におけるステップ S 1) が無いことである。これは、状態変数が関数の引数として渡されるデータの一要素となっているためである。

また、図 1 7 (b) のプログラム例と図 5 (b) のプログラム例との違いは、関数の引数としてデータテーブル 1 2 4 のアドレス値 (struct xxx*a) が渡されることと、状態変数がデータ構造体の一要素 (a.state) として持つことである。尚、“struct xxx*a” は xxx という構造体を持つ変数 a のアドレス値を表している。状態変数の更新は、データ構造体の一要素の更新が (a.state =) で示されている。

【 0 0 6 4 】

図 1 8 は、タスク呼出しを行うメインルーチン処理の説明図であり、(a) はそのフローチャート、(b) はその処理を C 言語で示した説明図である。

図 1 8 (a) と図 6 (a) との違いは、ステップ S 1 3 a のタスク番号に対応

したデータの取り出し処理、即ち、データテーブル 1 2 4 からタスク番号に対応したデータアドレスの取り出しが入るだけである。他のステップ S 1 1 a、S 1 2 a、S 1 4 a、S 1 5 a は、図 6 (a) におけるステップ S 1 1、S 1 2、S 1 4 と同様の処理である。

また、図 1 8 (b) のプログラム例と、図 6 (b) のプログラム例の違いは、配列関数 func の引数に *data [NUM] が入っているだけである。これは、データテーブル 1 2 4 の配列 data [] からタスク番号 NUM のデータアドレスを取り出すことに相当する。

以上、このようなタスク番号に対して、プログラムテーブル 1 2 1 とデータテーブル 1 2 4 とを分けてやれば、異なったタスク番号において、同じプログラムで異なったデータに対する処理が可能となる。

【 0 0 6 5 】

図 1 9 は、このような処理を行う場合のプログラムテーブル 1 2 1 とデータテーブル 1 2 4 の一例を示す説明図である。

図示のように、異なったタスク番号 (b111100 と b111010) において、同じプログラム (func1) で異なったデータ (data0, data1) に対する処理を行うことが可能となる。

【 0 0 6 6 】

〈効果〉

以上のように、具体例 4 によれば、任意のタスク番号に対応したプログラムアドレスとデータアドレスとを示すプログラムテーブル 1 2 1 とデータテーブル 1 2 4 とを設け、マトリクス回路 1 から特定のタスク番号を読み出した場合は、これらテーブルに示すアドレスに基づきタスクプログラムを実行し、かつ、データアクセスを行うようにしたので、例えば、同じデータ構成であるが異なったデータに対して同じ処理を行うような場合、データ個々にプログラムを設定する必要がなく、プログラムサイズが小さくて済む。これは、特にオブジェクト指向プログラムのようにデータとプログラムが一つの構造となったものに対して有効である。

【図面の簡単な説明】

【図 1】

本発明のタスクプログラム制御システムの具体例 1 を示す構成図である。

【図 2】

本発明の具体例 1 の全体的な構成図である。

【図 3】

典型的なプロトコルシーケンスの処理例の説明図である。

【図 4】

図 3 のプロトコルの処理を関数で表した説明図である。

【図 5】

本発明の具体例 1 で実行されるタスクプログラムの説明図である。

【図 6】

タスク呼出しを行うメインルーチン処理の説明図である。

【図 7】

タスクの初期割付状況の説明図である。

【図 8】

各タスクプログラムの説明図である。

【図 9】

メインメモリ上でのプログラムテーブルの割付状態を示す説明図である。

【図 1 0】

タスクの実行順序を示す説明図である。

【図 1 1】

具体例 2 の全体構成図である。

【図 1 2】

具体例 2 におけるタスクスケジューラの要部構成図である。

【図 1 3】

具体例 3 におけるタスクスケジューラの要部構成図である。

【図 1 4】

具体例 3 の動作の一例を示すための説明図である。

【図 1 5】

具体例 4 の全体構成図である。

【図 1 6】

具体例 4 におけるプログラムテーブルとデータテーブルの構成説明図である。

【図 1 7】

具体例 4 で実行されるタスクプログラムの構造を示す説明図である。

【図 1 8】

タスク呼出しを行うメインルーチン処理の説明図である。

【図 1 9】

具体例 4 におけるプログラムテーブルとデータテーブルの一例を示す説明図である。

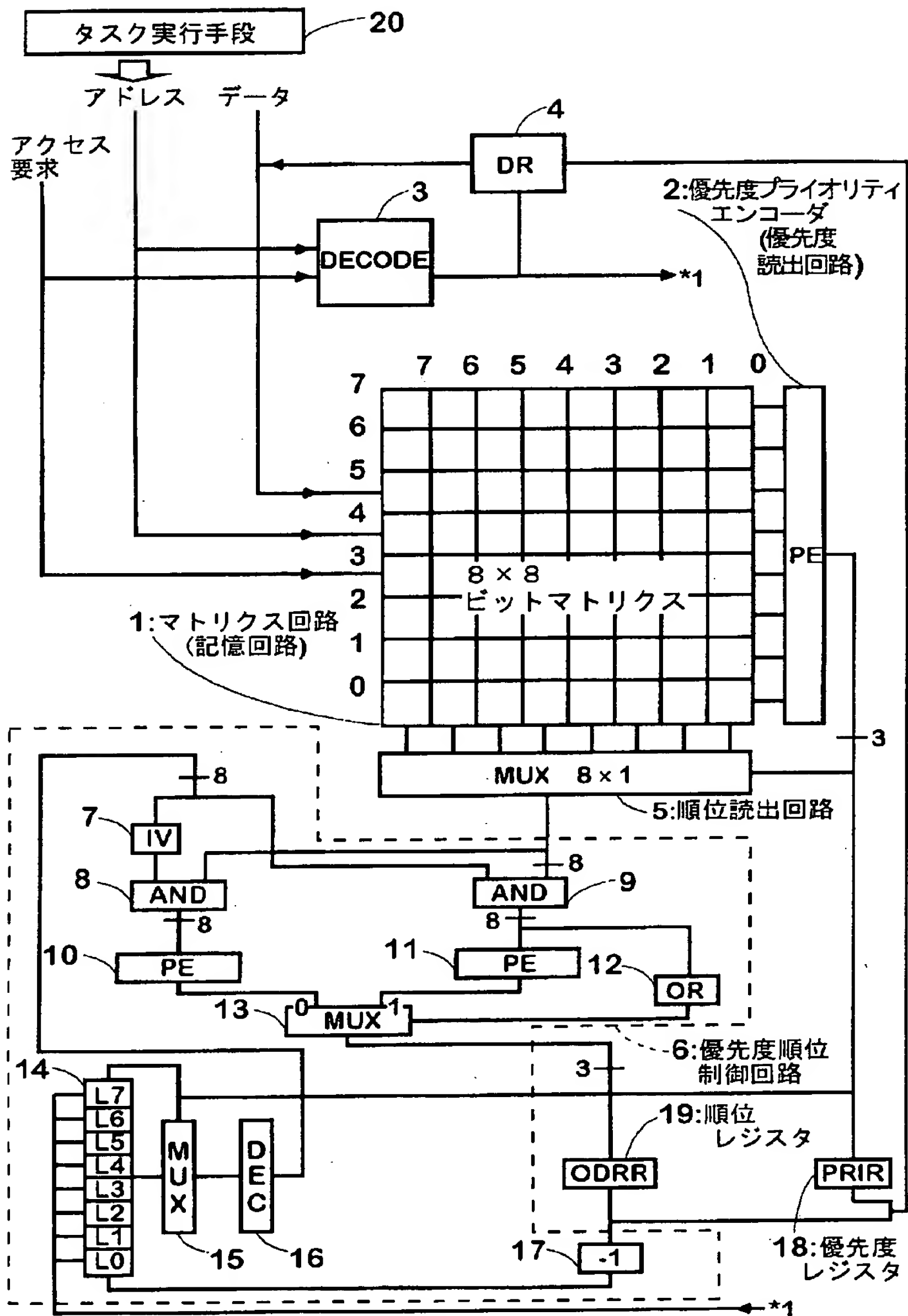
【符号の説明】

- 1 マトリクス回路（記憶回路）
- 2 優先度プライオリティエンコーダ（優先度読出回路）
- 5 マルチプレクサ（順位読出回路）
- 6 優先度別順位制御回路
- 1 8 優先度レジスタ
- 1 9 順位レジスタ
- 2 0 タスク実行手段
- 2 1 カウンタ
- 2 2 マスク回路
- 2 3 特定優先度読出回路
- 1 0 0、1 0 0 - 1、1 0 0 - 2 CPU
- 1 1 0、1 1 0 a タスクスケジューラ
- 1 2 0 メインメモリ
- 1 2 1 プログラムテーブル
- 1 2 2、1 2 2 a メインルーチン
- 1 2 3 各種タスクプログラム
- 1 2 4 データテーブル

特 2 0 0 0 - 2 7 0 9 2 5

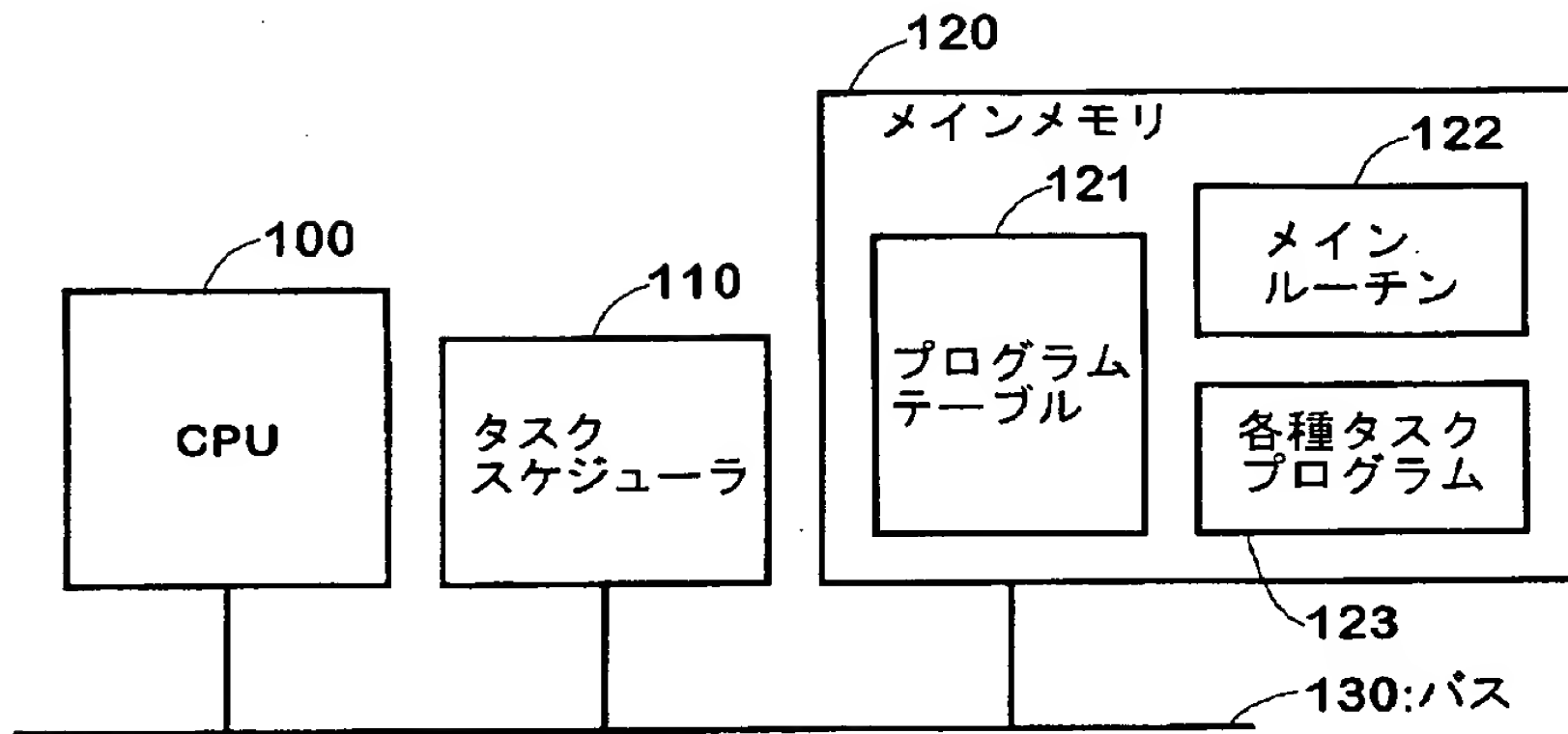
【書類名】 図面

【図 1】



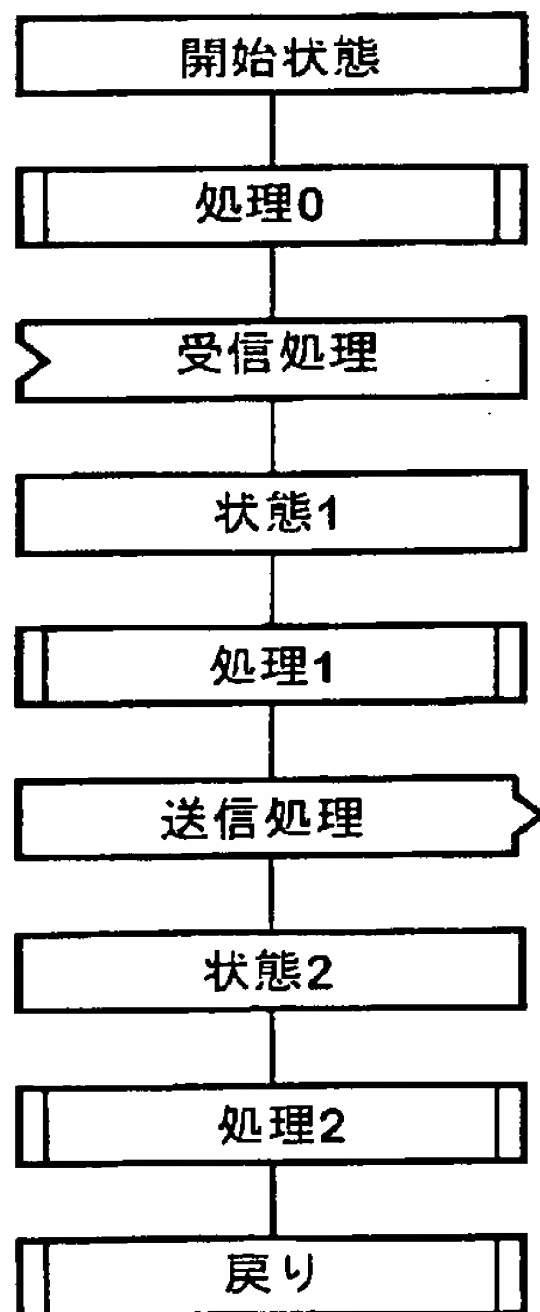
具体例 1 の構成図

【図 2】



具体例 1 の全体構成図

【図 3】



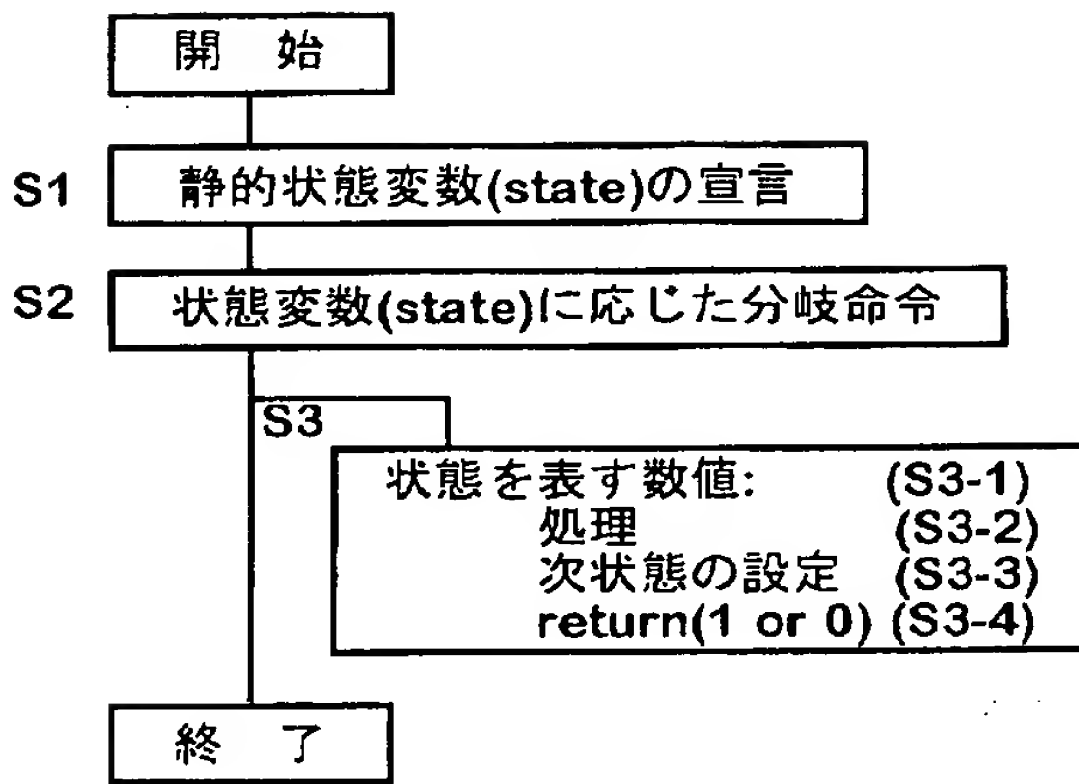
典型的なプロトコルシーケンスの処理例の説明図

【図 4】

```
func()
{
    proc0:
        処理内容0;
    recieve(chanel0, data);
    proc1:
        処理内容1;
    send(chanel0, data);
    proc2:
        処理内容2;
    return;
}
```

関数表示例の説明図

【図 5】



(a)

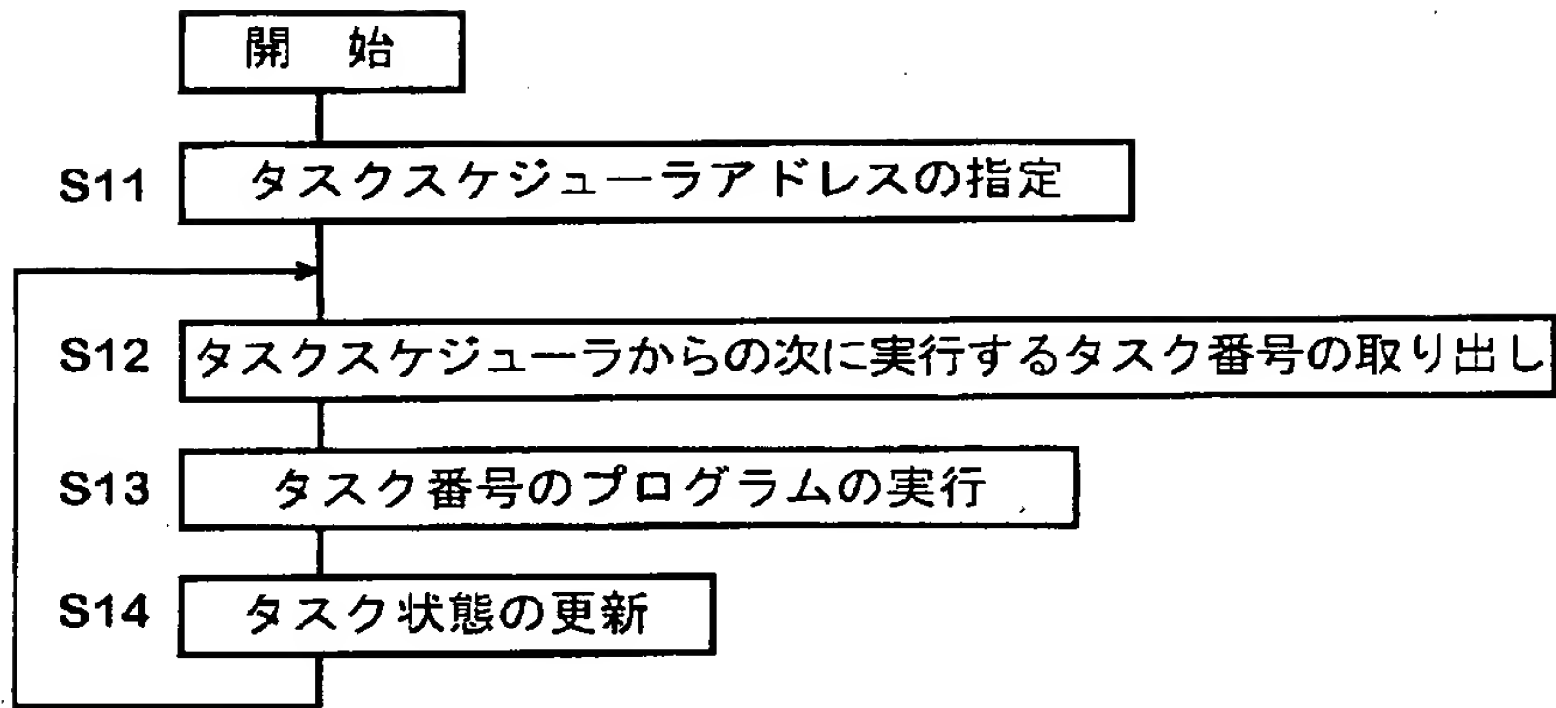
```

static int state; // S1
func()
{
    switch(state&0x3) { // S2
    //S3
    case0: //S3-1
        処理内容0; // S3-2
        state=1; // S3-3
        return(0); // S3-4
    case1:
        get(chanel0,data);
        処理内容1;
        send(chanel0, data);
        state=2;
        return(0);
    case2:
        処理内容2;
        state=0;
        return(0);
    defaults:
        state=0;
        return(0);
    }
}
  
```

(b)

具体例 1 で実行されるタスクプログラムの説明図

【図 6】



(a)

```

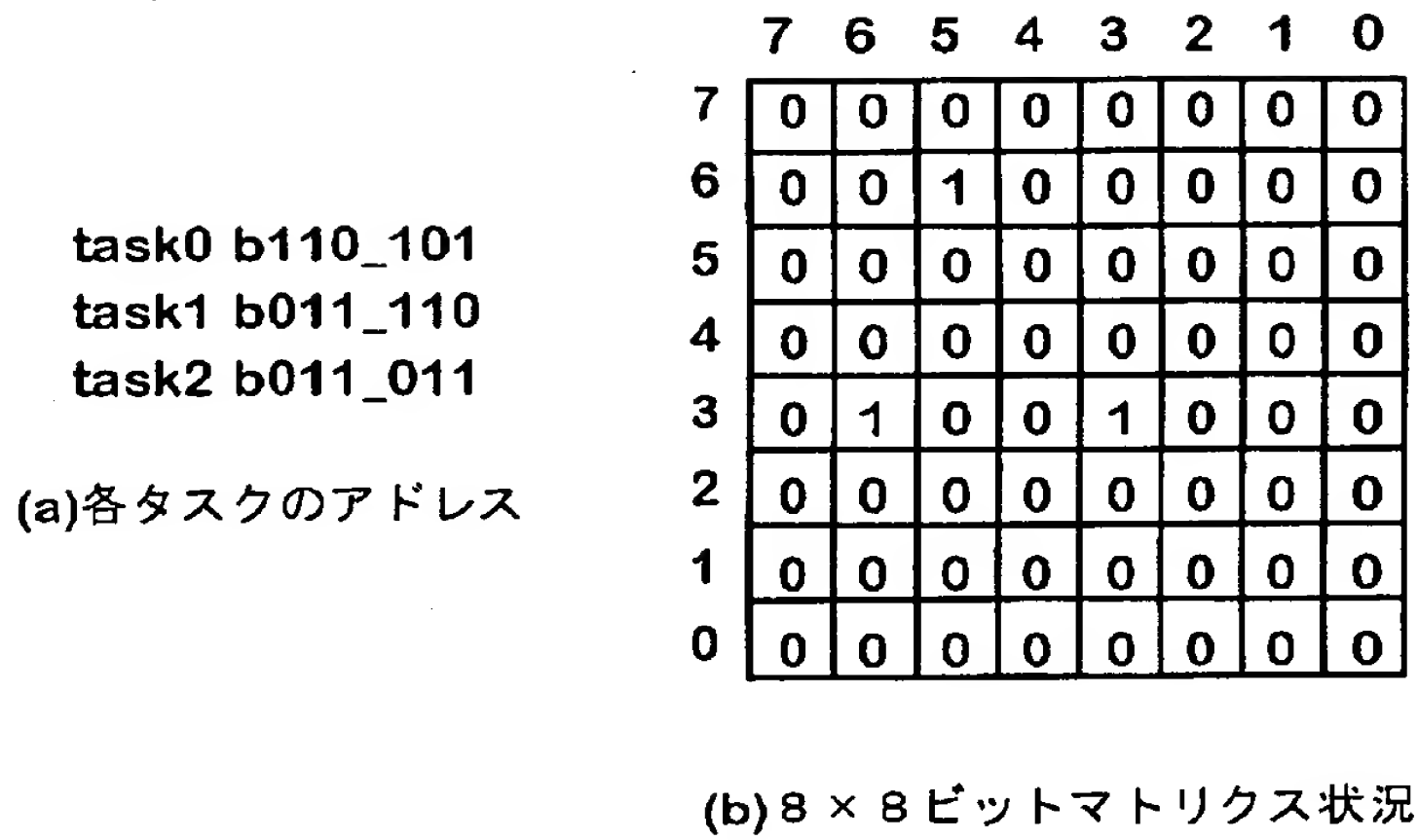
int. *sp;
sp= TASK_SCHEDULER_ADR; // S11

while(1) {
    NUM= *sp; // S12
    state[NUM]=func[NUM](); // S13&S14
}
    
```

(b)

メインルーチン処理の説明図

【図 7】



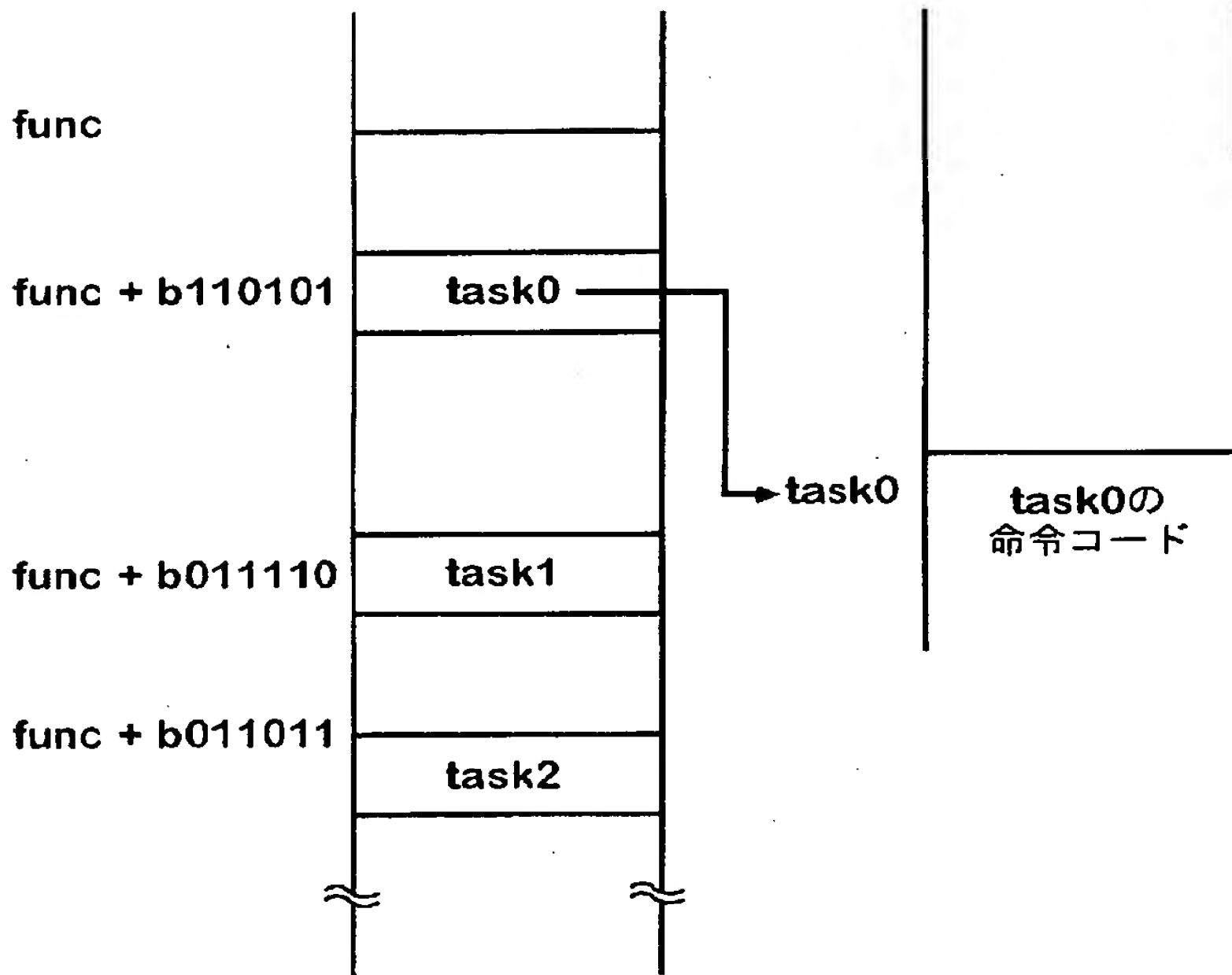
タスクの初期割付状況の説明図

【図 8】

<pre>static int sta0; int task0() { switch(sta0&0x1) { //13_S case0: sta0=1; return(0); case1: //30_S sta0=0; return(0); } }</pre>	(a)task0の内容
<pre>static int sta1; int task1() { switch(sta1&0x1) { //19_S case0: sta1=1; return(1); case1: sta1=0; return(0); } }</pre>	(b)task1の内容
<pre>static int sta2; int task2() { switch(sta2&0x1) { //26_S case0: state(task0_id)=1; sta2=1; return(1); case1: sta2=0; return(0); } }</pre>	(c)task2の内容

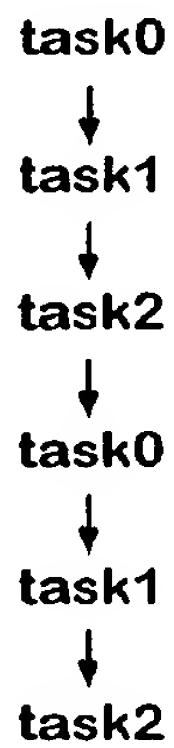
各タスクプログラムの説明図

【図 9】



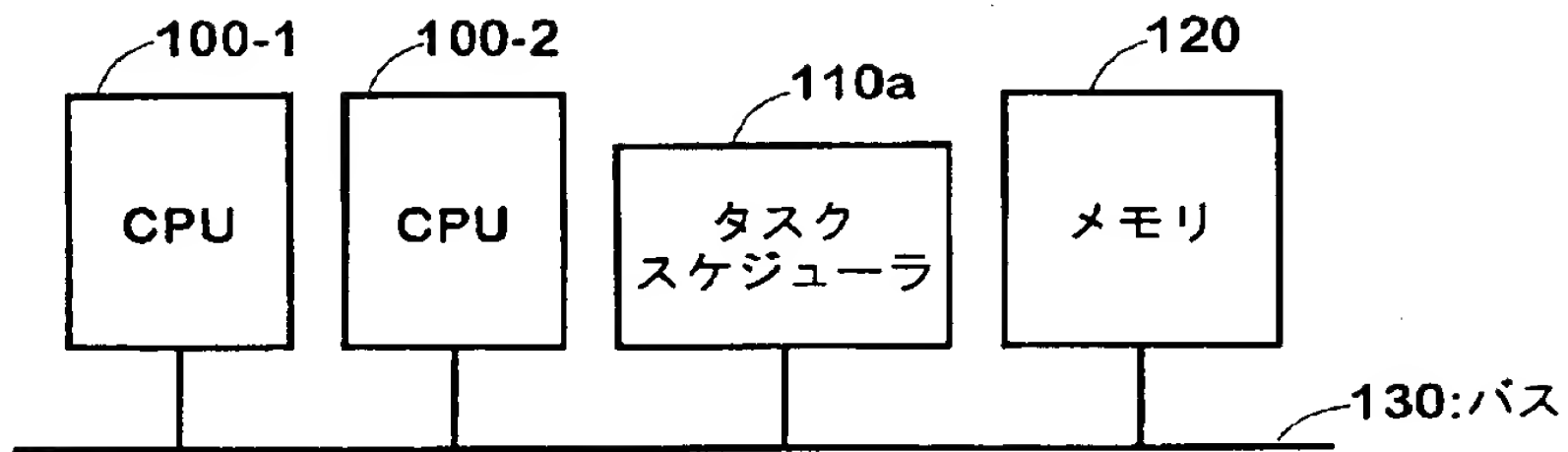
メモリ上でのプログラムテーブル割付状態の説明図

【図 1 0】



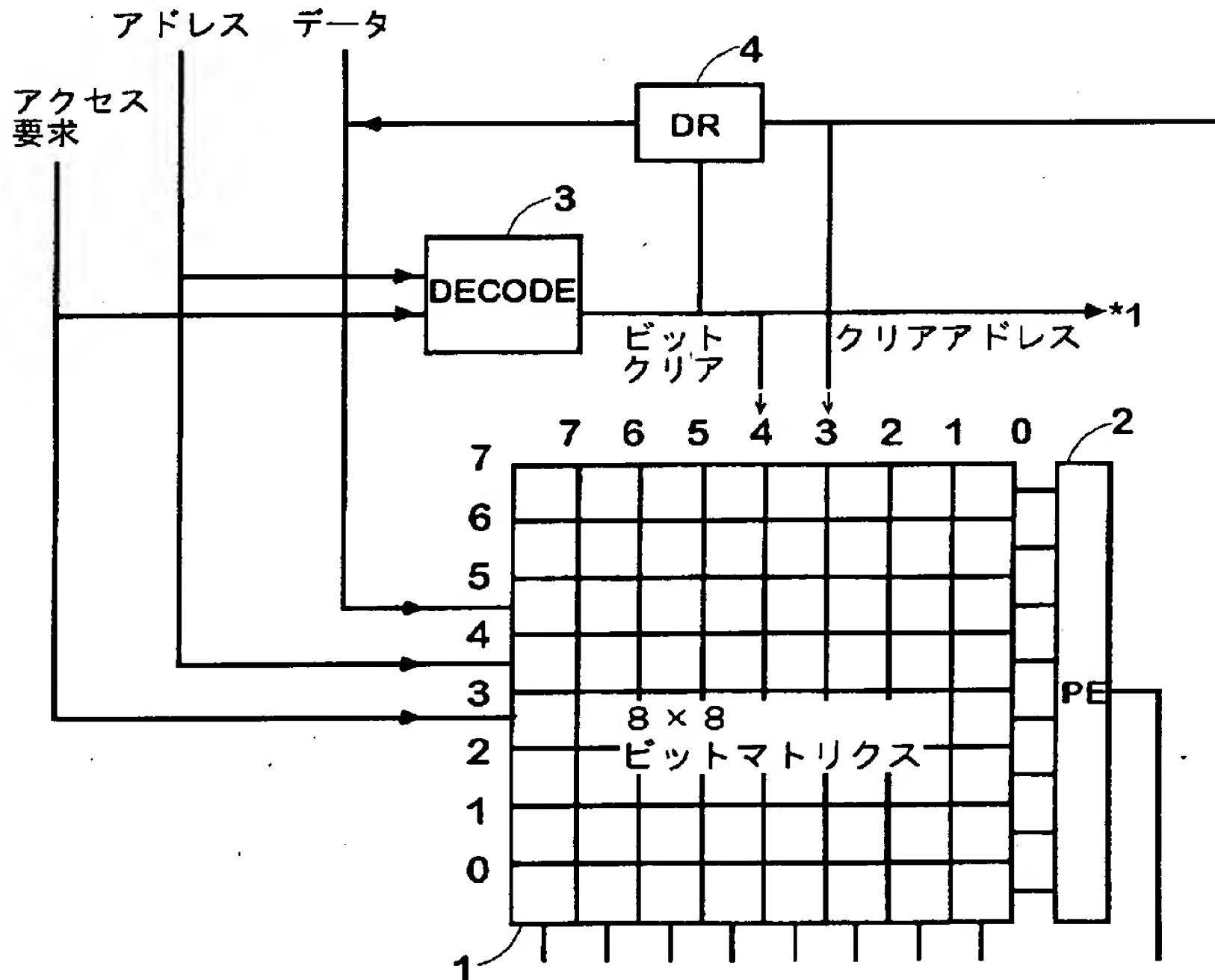
タスクの実行順序の説明図

【図 1 1】



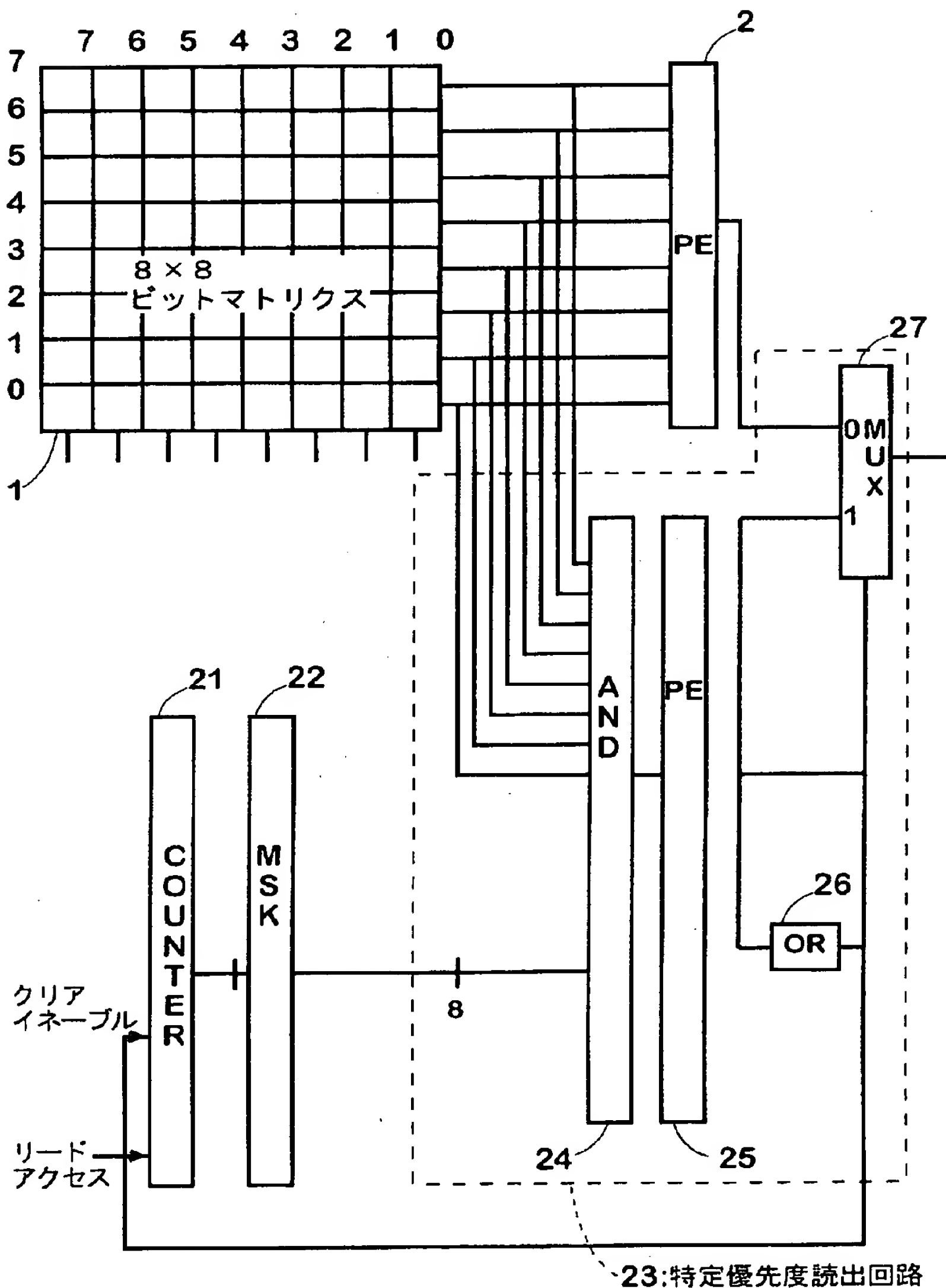
具体例 2 の全体構成図

【図 1 2】



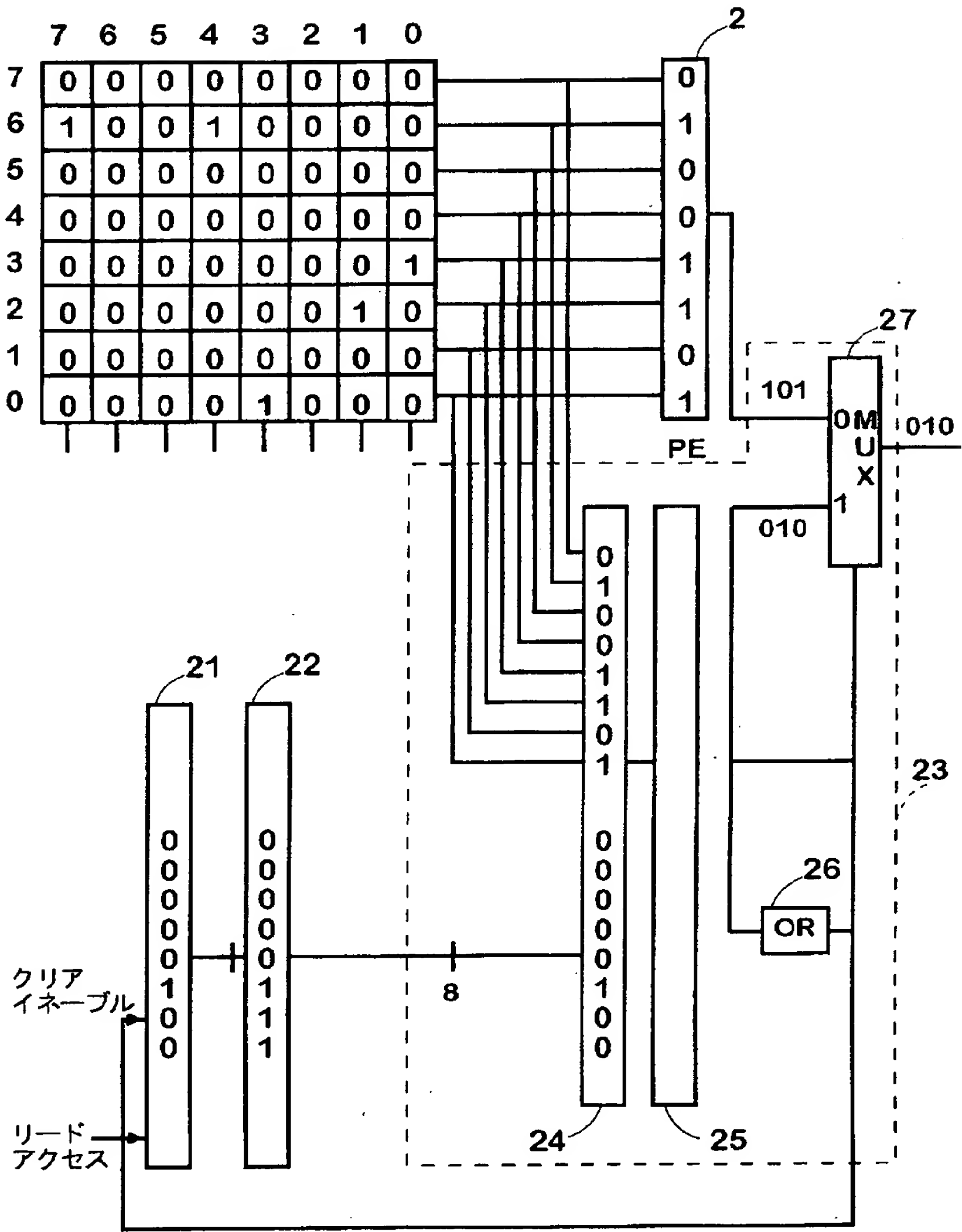
具体例 2 のタスクスケジューラの要部構成図

【図 13】



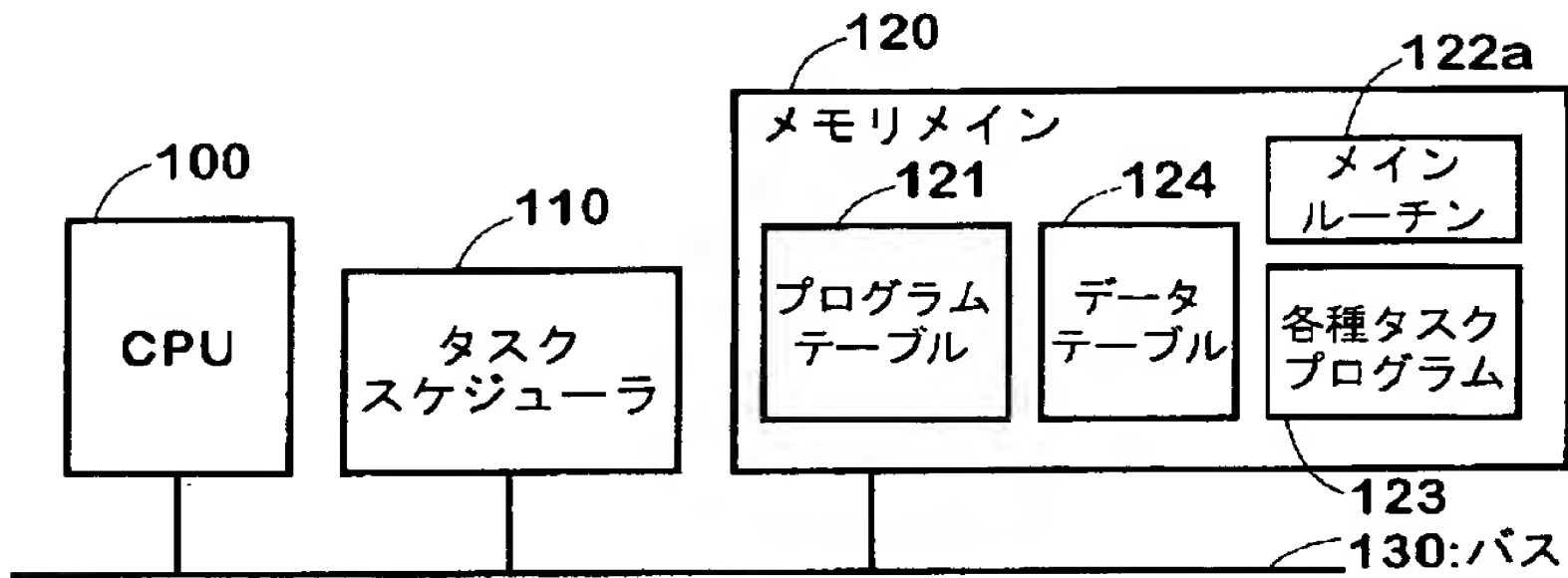
具体例 3 のタスクスケジューラの要部構成図

【図 1 4】



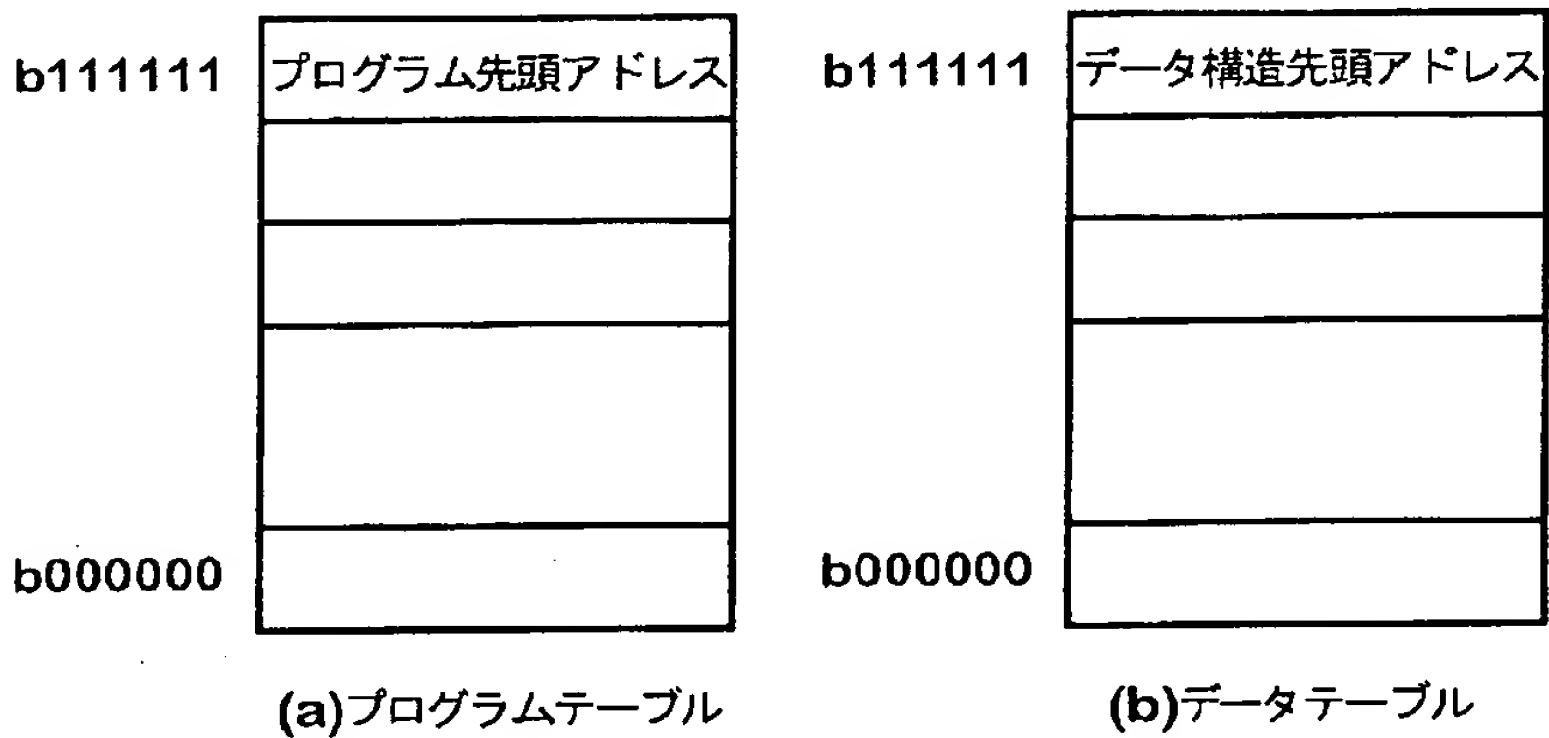
具体例 3 の動作の一例の説明図

【図 1 5】



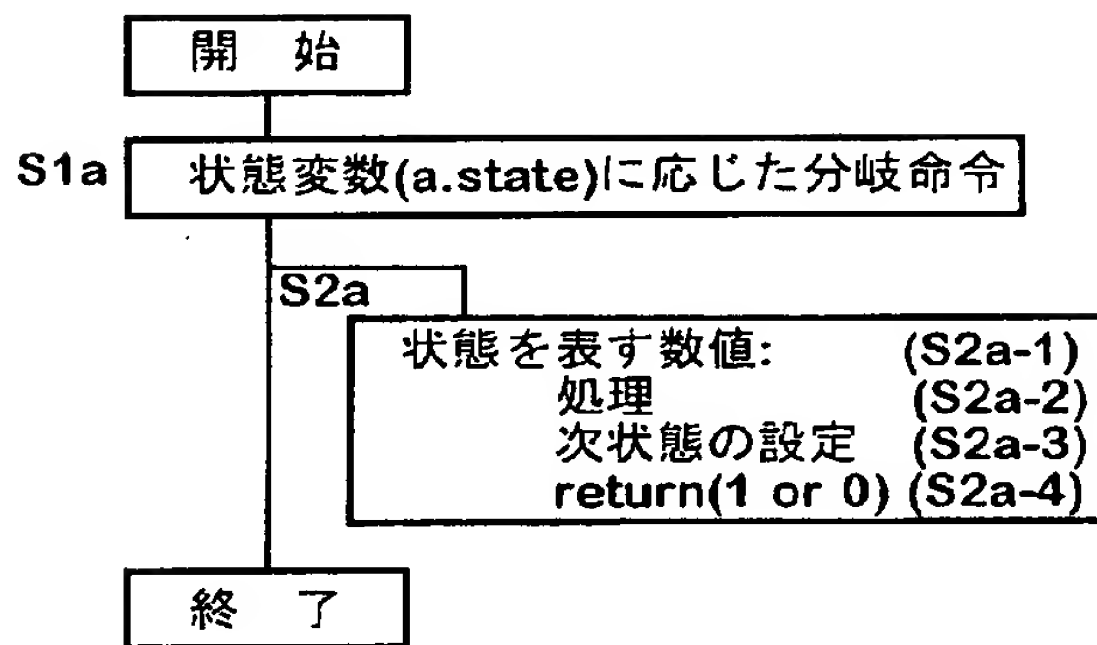
具体例 4 の全体構成図

【図 1 6】



具体例 4 のプログラムテーブルとデータテーブルの説明図

【図 1 7】



(a)

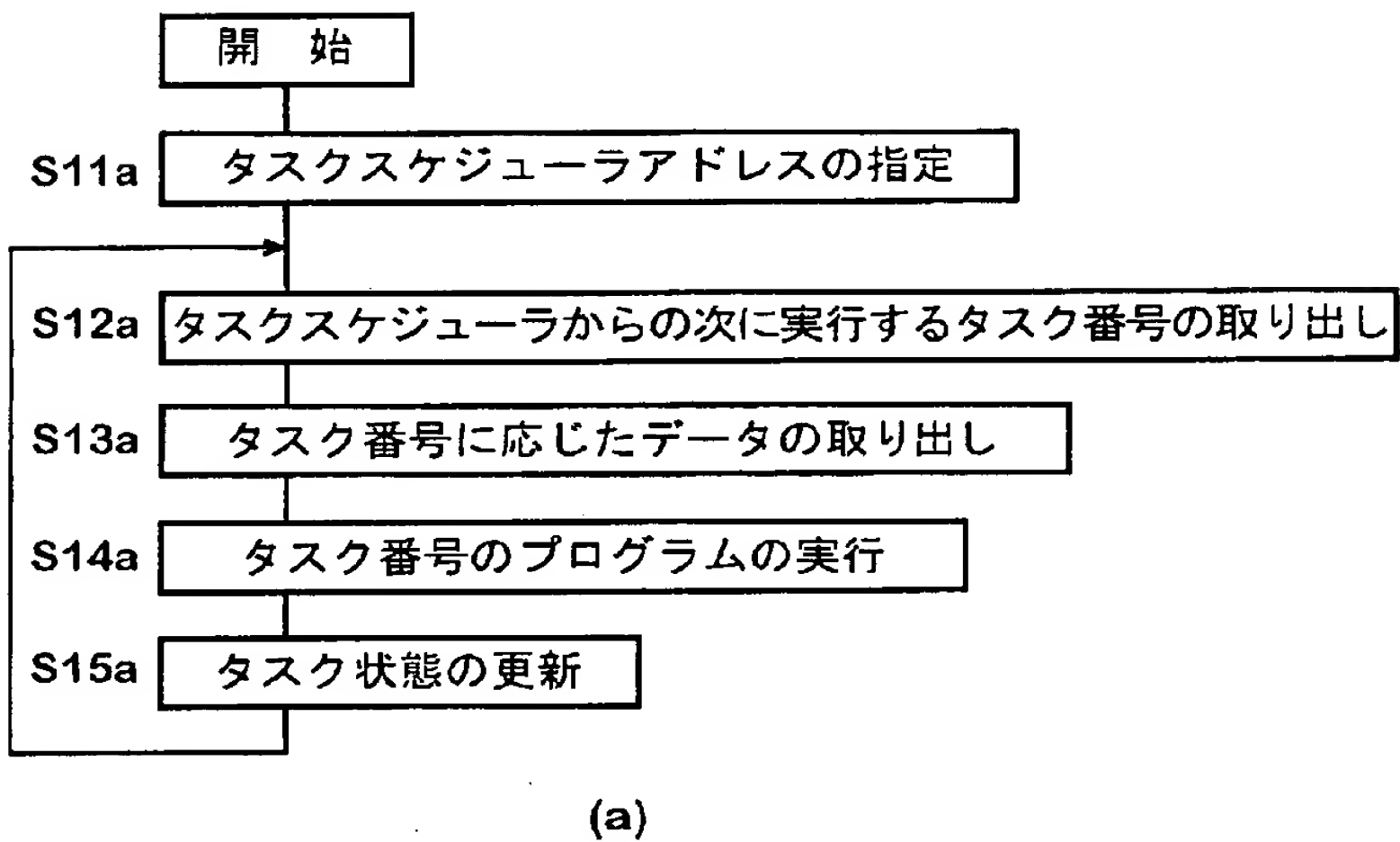
```

int. func(struct xxx*a)
{
    switch(a.state&0x3) { // S1a
    //S2a
    case0: //S2a-1
        処理内容0; // S2a-2
        state=1; // S2a-3
        return(0); // S2a-4
    case1:
        get(chanel0,data);
        処理内容1;
        send(chanel0, data);
        a.state=2;
        return(0);
    case2:
        処理内容2;
        a.state=0;
        return(0);
    defaults:
        a.state=0;
        return(0);
    }
}
  
```

(b)

具体例 4 で実行されるタスクプログラムの説明図

【図 1 8】



```

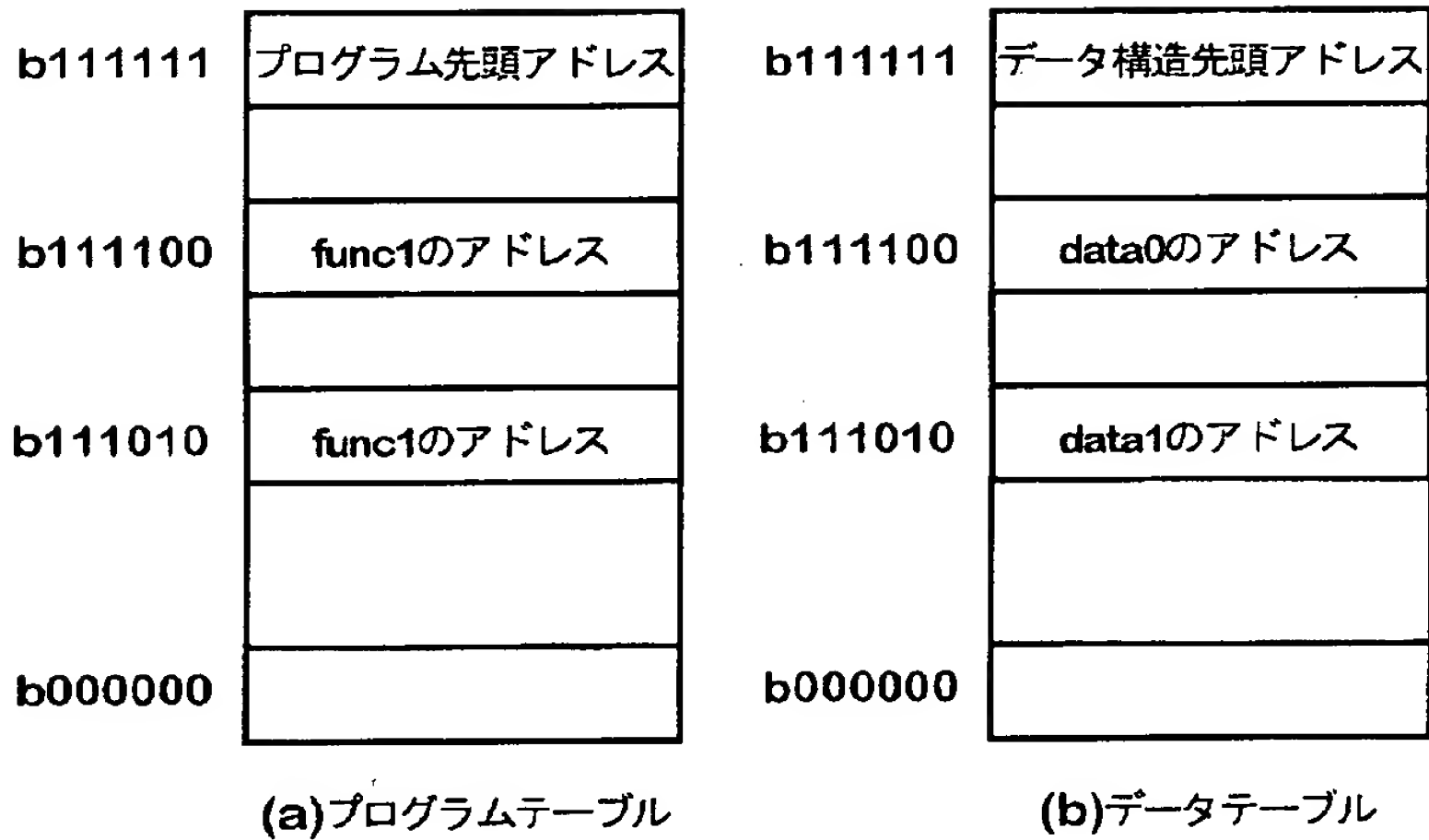
int. *sp;
int. *data[];
sp= TASK_SCHEDULER_ADR; // S11a

while(1) {
    NUM= *sp; // S12a
    state[NUM]=func[NUM](data[NUM]); // S14a&15a&S13a
}
  
```

(b)

具体例 4 のメインルーチン処理の説明図

【図 1 9】



具体例 4 のプログラムテーブルとデータテーブルの一例の説明図

【書類名】 要約書

【要約】

【課題】 複数のタスクの切替えを高速に行う。

【解決手段】 マトリクス回路 1 には複数のタスクの優先度と優先度別順位の情報が記憶されている。優先度プライオリティエンコーダ 2 は、マトリクス回路 1 から最も優先度の高い値を取り出す。この値は優先度レジスタ 1 8 に保持される。優先度別順位制御回路 6 は、優先度別順位のうち最も順位の高い値を順位レジスタ 1 9 に設定し、かつ、この値が次に読み出す場合は最も低い順位となるよう設定する。タスク実行手段 2 0 は、次に実行するタスクのアドレスを優先度レジスタ 1 8 および順位レジスタ 1 9 から読み出し、そのタスクを実行し、実行後は、タスクの状態に基づきマトリクス回路 1 の情報を更新する。

【選択図】 図 1

特 2 0 0 0 - 2 7 0 9 2 5

認定・付加情報

特許出願の番号	特願 2 0 0 0 - 2 7 0 9 2 5
受付番号	5 0 0 0 1 1 4 1 4 9 3
書類名	特許願
担当官	第七担当上席 0 0 9 6
作成日	平成 1 2 年 9 月 8 日

<認定情報・付加情報>

【提出日】 平成12年 9月 7日

次頁無

特2000-270925

出願人履歴情報

識別番号 [000000295]

1. 変更年月日 1990年 8月22日
[変更理由] 新規登録
住 所 東京都港区虎ノ門1丁目7番12号
氏 名 沖電気工業株式会社